|  |
| --- |
| Comp 3411 |
| Operating Systems |
| Final Exam Reference Book |

|  |
| --- |
| Ryan Woodward  8-18-2020 |

Contents

[Module 1: Introduction 1](#_Toc48822334)

[Learning Objectives 1](#_Toc48822335)

[Questions 3](#_Toc48822336)

[Module 2: Operating System Structures 4](#_Toc48822337)

[Learning Objectives 4](#_Toc48822338)

[Questions 5](#_Toc48822339)

[Module 3: Processes 8](#_Toc48822340)

[Learning Objectives 8](#_Toc48822341)

[Questions 10](#_Toc48822342)

# Module 1: Introduction

## Learning Objectives

* **Define what an operating system is.**
  + Software that manages a computer’s hardware. An intermediary between the computer user and the computer hardware.
  + A resource allocator; Manages the resources that are required to solve a problem.
  + A control program; Manages the execution of user programs to prevent errors and improper use of the computer.
  + The one program always running on a computer, called the kernel.
    - Also 2 other running programs
      * System programs which are associated with the os but not necessarily a part of the kernel.
      * Application programs. Not involved with running of the system.
  + Middleware framework that eases application development
* **List operating systems of different types.**
  + IOS, Android, Windows, OS X, Linux
* **Define what multiprogramming, parallel processing, and time sharing are.**
  + Multiprogramming is where multiple processes can be run at the same time. There is a subset of jobs in memory, and a scheduler will determine which process gets executed at what time.
  + Multiple processors with multiple cores can operate on jobs simultaneously
  + Time sharing is where each process will execute for a certain amount of time on the CPU, once its time is up the scheduler will swap a new process in if the previous one has been determined to of a lower priority or has finished executing
* **Define what real-time systems and embedded systems are.**
  + Real-time systems are designed to run with user intervention, whereas for the most part embedded are designed to run without user intervention.
* **Explain how an operating system starts when the user turns on the computer system.** 
  + The bootstrap program runs when the computer is turned on. It loads and starts the OS.
* **Define what bootstrap is.**
  + The first program that is run when a computer is turned on. Stored in the EEPROM. Loads the OS
* **Explain how an interrupt can be handled.**
  + CPU hardware monitors a wire called the interrupt-request line. This is a hardware connection.
    - Most CPUs have 2. One is unmaskable and one is maskable.
      * Nonmaskable cannot be delayed or blocked. Reserved for specific events.
      * Maskable can be delayed or blocked
      * This provides interrupt priority levels
  + CPU detects signal, reads interrupt number and looks up that number in the interrupt vector
  + Interrupt vector points to an interrupt handler routine chain, where each interrupt handler routine is called one-by-one until one is found that can service the request.
  + Interrupt handler saves any state it will be changing, determines cause of interrupt, performs the processing, performs a state restore, and returns a *return\_from\_interrupt* instruction to return the CPU to the execution state prior to the interrupt.
  + Device controller raises interrupt on the interrupt request line, CPU catches the interrupt and dispatches it to the interrupt handler, handler clears the interrupt by servicing the device.
* **Explain how an interrupted program can resume later.**
  + Interrupt handler saves any state it will be changing, determines cause of interrupt, performs the processing, performs a state restore, and returns a *return\_from\_interrupt* instruction to return the CPU to the execution state prior to the interrupt.
* **List storage types in the ascending order of access time.**
  + Registers, cache, RAM, Secondary (electrical), Secondary (mechanical), Tertiary
* **List the four types of hardware protection.**
  + Dual mode, cpu, i/o, memory
* **Explain how memory protection and CPU protection can be supported.**
  + Memory protection is supported by paging and access levels. CPU protection is supported by a timer.
* **Discuss how operating systems are used in various computing environments**
  + Virtualization allows different OS’s to be run on different machines. Thus, an emulated OS can be run. However, this slows down the performance as each source instruction needs to be translated to a native machine instructions. \Often resulting in many more instructions and a performance hit.
  + These emulated operating systems are handled by a virtual machine manager
    - In some cases, these VMMs are becoming the base OS
* **Provide examples of free and open-source operating systems.**
  + Free
    - Unix
  + Open Source
    - Linux distros
    - Bsd unix
    - solaris

## Questions

**1.1**

The 3 main purposes of an operating system are: Resource Allocator, controlling processes, intermediary between the user and the hardware (environment for the user to be able to run applications and not have to deal with the complexities of the hardware).

**1.2**

*We have stressed the need for an operating system to make efficient use of the computing hardware. When is it appropriate for the operating system to forsake this principle and to “waste” resources? Why is such a system not wasteful?*

Single-user systems should maximize the use of the system for the user. A GUI might “waste” CPU cycles, but it optimizes the user’s interaction with the system

**1.5**

*How does the distinction between kernel mode and user mode function as a rudimentary form of protection?*

This prevents the user from using privileged instructions. The kernel mode has specialized instructions that control i/o, memory, and cpu access that a user could mess up.

**1.6**

*Which of the following instructions should be privileged?*

1. *Set value of a timer*
   1. Privileged
2. *Read the clock*
   1. Non
3. *Clear memory*
   1. Privileged
4. *Issue a trap instruction*
   1. Non
5. *Turn off interrupts*
   1. Privileged
6. *Modify entries in device status table*
   1. privileged
7. *Switch from user to kernel mode*
   1. Non (but the book says it’s privileged)
8. *Access i/o device*
   1. Privileged

**1.7**

*Some early computers protected the operating system by placing it in a memory partition that could not be modified by either the user job or the operating system itself. Describe two difficulties that you think could arise with such a scheme.*

The data required by the OS (passwords, access controls, accounting info, etc.) would have to be passed through unprotected memory and thus be accessible to unauthorized users.

**1.8**

*Some CPUs provide for more than two modes of operation. What are two possible uses of these multiple modes?*

Could have an I/O mode that would be a quick check to determine what state a process is currently in. Could have a mode that locks everyone else out until it’s finished doing its operations.

**1.9**

*Timers could be used to compute the current time. Provide a short description of how this could be accomplished.*

Set a timer for the future and go to sleep. When awakened by the interrupt, update its local state, which it uses to keep track of the number of interrupts it has received thus far. Repeat this process of continually setting timer interrupts and updating its local state when the interrupts are raised.

# Module 2: Operating System Structures

## Learning Objectives

* List the five core components.
  + Kernel, API, User interface and file system, hardware devices, device drivers
* Define what a process is.
  + A process is a program that is executing
* List the five core services.
  + User Interface, program execution, i/o operations, file system manipulation, communications
* Define what a system call is.
  + A request from the user to change into kernel mode and user privileged kernel instructions
  + *A programming interface to the services provided by the OS*
* Define what a system program is.
  + An environment for program development and execution
* Define what an OS kernel is.
  + The software that is running all the time on your computer for your OS
  + Consists of everything below the system call interface and above the hardware
  + Provides the file system, cpu scheduling, memory management, and other os functions
* Describe the services an operating system provides to: users, processes, and other systems.
  + Users
    - User interface
    - Program execution
    - i/o operations
    - file system manipulation
    - communications
    - error detection
  + processes
    - resource allocation
    - logging
    - scheduling
    - interprocess communication
  + other systems
    - protection and security
    - communication
* Discuss the various ways of structuring an operating system.
  + Monolithic kernel
  + Microkernel
  + Layered kernel
  + Hybrid systems
  + Modules
* Explain how operating systems are installed and customized and how they boot.
  + Installation
    - Configure the kernel and place into a .config file
    - Compile the kernel based on the .config file
    - Compile the kernel modules
    - Install the modules
    - Install the kernel on the system, where it will be placed in a nonvolatile memory location. Known as the boot block
  + Booting
    - The bootstrap program locates kernel
      * The bios contains an address which points towards the boot block which contains the kernel.
    - Kernel is loaded into memory
    - Kernel initializes the hardware
    - The rootsystem is mounted

## Questions

**Questions 2.1 and 2.3–2.11**

**2.1**: What is the purpose of system calls?

System calls allow user level processes to request services of the operating system

**2.3:** What system calls must be executed by a command interpreter or shell in order to start a new process on a UNIX system?

A fork() system call and an exec() system call need to be performed to start a new process. The fork() call clones the currently executing process, while the exec() call overlays a new process based on a different executable over the calling process

**2.4** What is the purpose of system programs?

System programs can be thought of as bundles of useful system calls. They provide basic functionality to users so that users do not need to write their own programs to solve common problems.

**2.5**: What is the main advantage of the layered approach to system design? What are the disadvantages of the layered approach?

As in all cases of modular design, designing an operating system in a modular way has several advantages. The system is easier to debug and modify because changes affect only limited sections of the system rather than touching all sections. Information is kept only where it is needed and is accessible only within a defined and restricted area, so any bugs affecting that data must be limited to a specific module or layer. The primary disadvantage to the layered approach is poor performance due to the overhead of traversing through the different layers to obtain a service provided by the operating system.

**2.6:** List 5 services provided by an operating system and explain how each creates convenience for users. In which cases would it be impossible for user-level programs to provide these services?

1. **Program execution**
   1. The operating system loads the contents (or sections) of a file into memory and begins its execution. A user-level program could not be trusted to properly allocate CPU time.
2. **I/O Operations**
   1. It is necessary to communicate with disks, tapes, and other devices at a very low level. The user need only specify the device and the operation to perform on it, and the system converts that request into device or controller specific commands. User level programs cannot be trusted to access only devices they should have access to and to access them only when they are otherwise unused.
3. **File system manipulation**
   1. There are many details in file creation, deletion, allocation, and naming that users should not have to perform. Blocks of disk space are used by files and must be tracked. Deleting a file requires removing the name file information and freeing the allocated blocks. Protections must also be checked to assure proper file access. User programs could either ensure adherence to protection methods nor be trusted to allocate only free blocks and deallocate blocks on file deletion
4. **Communications**
   1. Message passing between systems requires messages to be turned into packets of information, sent to the network controller, transmitted across a communications medium, and reassembled by the destination system. Packet ordering and data correction must take place. Again, user programs might not coordinate access to the network device, or they might receive packets destined for other processes.
5. **Error detection**
   1. Error detection occurs at both the hardware and software levels. At the hardware level, all data transfers must be inspected to ensure that data have not been corrupted in transit. All data on media must be checked to be sure they have not changed since they were written to the media. At the software level, media must be checked for data consistency. For instance, whether the number of allocated and unallocated blocks of storage match the total number on the device. There, errors are frequently process independent (for instance, the corruption of data on a disk), so there must be a global program (the operating system) that handles all types of errors. Also, when errors are processed by the operating system. Processes need not contain code to catch and correct all the errors possible on a system.

**2.7:** Why do some systems store the operating system in firmware, while others store it on disk?

For certain devices, such as embedded systems, a disk with a file system may not be available for the device. In this in this situation, the operating system must be stored in firmware.

**2.8:** How could a system be designed to allow a choice of operating systems from which to boot? What would the bootstrap program need to do?

Consider a system that would like to run both Windows and three different distributions of Linux (for example, red hat, Debian, and ubuntu). Each operating system will be stored on disk. During system boot, a special program (which we will call the boot manager) will determine which operating system to boot into. This means that rather than initially booting to an operating system, the boot manager will first run during system Startup. It is this boot manager that is responsible for determining which system to boot into. Typically, not managers must be stored at certain locations on the hard disk to be recognized during system Startup. Boot managers often provide the user with a selection of systems to boot into; boot managers are also typically designed to boot into a default operating system if no choice is selected by the user.

**2.9:** The services and functions provided by an operating system can be divided into two main categories. Briefly describe the two categories and discuss how they differ.

One class of services provided by an operating system is to enforce protection between different processes running concurrently in the system. Processes can access only those memory locations that are associated with their address spaces. Also, processes are not allowed to corrupt files associated with other users. A process is also not allowed to access devices directly without operating system intervention. The second class of services provided by an operating system is to provide new functionality that is not supported directly by the underlying hardware. Virtual memory and file systems are two such examples of new services provided by an operating system.

**2.10:** Describe three general methods for passing parameters to the operating system

1. Pass the parameter in registers
   1. This may prove insufficient when there are more parameters than registers
2. Store the parameters in a block, or table, in memory, and pass the address of the block as a parameter in a register
   1. This approach is used by Linux and Solaris
3. Push the parameters onto a stack; to be popped off by the OS.
   1. Block and stack methods do not limit the number or length of parameters passed.

**2.11:** Describe how you could obtain a statistical profile of the amount of time a program spends executing different sections of its code. Discuss the importance of obtaining such a statistical profile.

Profiling is a form of dynamic program analysis which serves to aid program optimization. It uses a wide variety of techniques to collect data such as time complexity of a program or the frequency and duration of function calls. Some examples of methods profiling use to collect these information’s are:

* Hardware interrupts
* Code instrumentation
* Performance counters
* Instruction set simulation

Typically, nowadays personal computers have a system profiler program that can provide detailed information about its software and hardware. In order to obtain a statistical profile of the amount of time spent by a program executing different sections of its code we would use from the methods describe above hardware interrupts. So, we would issue periodic timer interrupts and when the interrupts are delivered, we would monitor and supervise what code blocks are being executed. As a result, we would get a profiling system which would monitor the code which is being executed which later could be used by the engineer to optimize the code blocks which are consuming a lot of CPU resources.

# Module 3: Processes

## Learning Objectives

* Identify five components composing a process.
  + Text – the executable code
  + Data – global variables
  + Heap – memory that is dynamically allocated during program run time
  + Stack – temporary data storage when invoking functions
  + Fifth ?
* Identify the states in which a process can be.
  + New – the process is being created
  + Running – instructions are being executed
  + Waiting – the process is waiting for some event to occur
  + Ready – the process is waiting to be assigned to a processor
  + Terminated – the process has finished execution
* Identify the seven information types stored in a PCB.
  + Process state
  + Program counter – the counter indicates the address of the next instruction to be executed for this process
  + Cpu registers
  + Cpu scheduling info – includes process priority, pointers to scheduling queues, and any other scheduling parameters
  + Memory management info – info includes value of the base and limit registers and page tables, or the segment tables depending on the memory system
  + i/o status information
* Explain how CPU switches from a process to another process.
  + Interrupt occurs and the kernel saves the current context of process the pcb
  + State restore is performed on the next priority process
    - This is pure overhead
* Define what the concept of multi-programming is.
  + Multiple processes running at the same time on one processor. This occurs by causing interrupts at scheduled intervals and allowing all processes a time quantum of the cpu
* State the reason why process scheduling is required.
  + Ensures that processes execute efficiently and have reduced wait times.
* Define what context switch is.
  + Context switch is when the kernel will save the state of a process currently executing on the cpu into the process control block after a time quantum and completed. It then will load the next highest priority process into the processor
* List the two core operations on processes.
  + Process creation
  + Process termination
* State with an example the reason why multiple processes need to cooperate.
  + Dividing processes into subtasks can lead to less complications and better modularity. These subtasks can be assigned to threads and performed in parallel if need be.
* List the two types of interprocess communication mechanisms.
  + Shared memory
  + Message passing
* Design programs that use pipes and POSIX shared memory to perform interprocess communication.
  + Shared memory
    - See pg 127. It’s basically just produced block looking to ensure that the section of memory doesn’t already contain something. If in == out this means that the buffer is empty, otherwise in will be greater than out, and out will point at consumable items
  + Message passing
    - See pg 137. Message passing involves sending messages to the kernel, and the kernel will alert the intended message
* Describe client-server communication using sockets and remote procedure calls.
  + Sockets
    - Can be on the same machine or a different machine
    - Consists of an ip address and a port number
    - Low level as it’s only an unstructured byte stream being transferred
  + Remote procedure
    - Known as subroutine or function call (RPC – remote procedure call)
    - See pg 148
    - Function call is sent to a server, server processes the call and returns the information
* Design Kernel modules that interact with the Linux operating system.
  + Search online

## Questions

3.1 Using the program shown in Figure 3.30, explain what the output will be at LINE A.

* Value = 5

3.2 Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

* 8 total processes running including the original

3.3 Original versions of Apple's mobile iOS operating system provided no means of concurrent processing. Discuss three major complications that concurrent processing adds to an operating system.

* Cpu scheduler must be aware of the concurrent processes and choose an appropriate algorithm that schedules the concurrent process
* Concurrent processes may need to communicate with each other, meaning a method for inter-process communication must be developed for concurrent processes
* Since mobile devices have limited memory, a process that handles memory poorly will have a negative impact on all other concurrent processes.

# Module 4: Threads

## Learning Objectives

* Identify the basic components of a thread, and contract threads and processes
  + Thread components
    - Thread ID, program counter, register set, and stack
  + Shared with other threads in the same process
    - Code section, data section, and other os resources such as open files and signals
  + Not sure what contract threads are
* Describe the major benefits and significant challenges of designing multithreaded processes.
  + Benefits
    - Resource sharing - (memory, data, address) makes sure resources are being used efficiently. This makes it more economical to use threads as opposed to multiple processes.
    - Responsiveness – If a program is waiting for something (user input, data, etc.) a different thread on the same program can perform operations that aren’t reliant on that data. An example is one thread performing user data, and one thread loading an image
    - Utilization of multiprocessor architecture – each thread can run on a different cpu, meaning that a program can run in parallel. This increases the concurrency of the system.
  + Challenges
    - Increase complexity
    - Complications due to concurrency
    - Difficult to identify errors
    - Testing complications – unable to determine things due to timing issues. Atomic calculations and such
    - Unpredictable results – once again atomic operations can lead to unpredictability
    - Complications of porting existing code
* Illustrate different approaches to implicit threading, including thread pools, fork-join, and Grand Central Dispatch
  + Thread Pools
    - Instead of starting a new thread for every task to execute concurrently, the task can be passed to a thread pool. As soon as the pool has any idle threads the task is assigned to one of them and executed. Internally the tasks are inserted into a Blocking Queue which the threads in the pool are dequeuing from
      * A blocking queue is a queue that blocks when you try to dequeue from it and the queue is empty, or if you try to enqueue items to it and the queue is already full. A thread trying to dequeue from an empty queue is blocked until some other thread inserts an item into the queue.
  + Fork-Join